The Design and Usage Model of LLNL Visualization Clusters

Lawrence Livermore National Laboratory

Dale A. Southard Jr.

Lawrence Livermore National Laboratory

21 February 2008

Who is Dale?

For the purposes of this talk, I'm the PPPE (pre and post-processing environment) vis system hardware architect.

I wear several other hats — SGI Sysadmin/PLG group member, PPPE developer, AISSO, and a few other things.

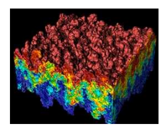
Prior to LLNL, I have been:

- Faculty with the Notre Dame College of Science
- Computational Chemistry Grad Student at University of Toledo
- Synthetic Organic Chemist
- AFF/SL/Tandem Skydiving Instructor

Today's Topics

1. Some basics
2. Brief history of LLNL vis architectures
3. Architecture and usage model of the current vis clusters
4. Future directions and security concerns

Vis Basics — Big Data



Visualization of large-scale data is inherently interactive (man-in-the-loop) and so the design of systems to do it is more "capability" than "capacity". As an example the Miranda dataset:

- 130 TB in size (so fitting the whole dataset in-core isn't possible).
- Problem has 27B zones with 5 floats/zone (so no existing output device could output a static picture that captured all the data)

So below a minimum capability, interactive performance drops to unacceptably low levels and working with the dataset is no longer feasible.

Vis Basics — The Four Paradigms

There are four approaches to handling data too large for a workstation:

Move the Data works well for relatively small timesteps and relatively fast networks. Common example is NFS.

Move the Triangles works well for relatively small timesteps and relatively fast networks. Common example is GLX.

Move the Pixels works well for large or complex timesteps. No *common* example, but VisIt, Paraview, and EnSight all support this mode of operation.

Move the RGB works well for large budgets and short distances. LLNL uses this model for driving PowerWalls.

Ancient History — IRIX-based Vis Systems

Before there were vis clusters, we had multi-processor ccNUMA systems with multiple graphics pipes and local fibre-channel arrays.

- Excellent single-threaded I/O rates
- Simple programming model
- One system to rule them all

but

- Large machines are expensive to procure and support
- Data movement problem

Transition to Clustered Linux Vis

In 2000, LLNL began pursuing clustered visualization using COTS as a strategy for replacement of the SGI systems.

First Generation custom-built cluster consisting of racked desktops running a vis-developed software stack. Good proof-of concept, but inadequate support and not enough I/O.

Second Generation vis clusters built by scaling down existing compute clusters and adding video cards. Support model improved, but performance inadequate due to lack of I/O bandwidth.

Third Generation custom-built vis clusters designed to use the same software stack as compute clusters.

Need for a fourth generation?

Today — Clustered Visualization

Large vis clusters like Gauss and Prism are designed to provide the best possible visualization performance while still running the LLNL CHAOS Linux stack.

- Affordable procurement and support model
- Leverages true commodity parts
- *In situ* data access for computational following/steering

Smaller vis clusters (Boole, Grant, Moebius, Stagg, Vertex) follow a simplified architecture plan and are used to drive PowerWalls.

LLNL Vis Cluster Architecture

- Software
- Support model
- I/O Performance
- Job management and security
- Does all this work

Software

Most of the important vis tools and techniques can utilize clustered resources.

- Vis/PP tools including VisIt provide a distributed rendering mode that uses a client-server backend to harness cluster resources
- Distributed rendering hooks have been integrated into the VTK toolkit
- Unmodified tools can use Chromium for sort-first and sort-last distributed rendering
- Future middle-ware technologies like Chromium Render Server will continue to extend the reach of clustered vis
- PowerWalls can utilize DMX (and sometimes Chromium) to run most apps on multi-node, multi screen walls without modification.

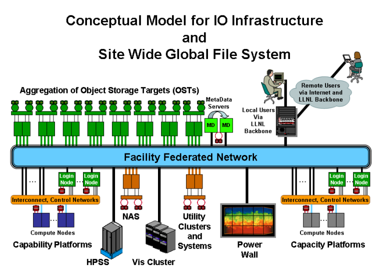
Support Model

After our experiences with the first generation vis cluster, we've worked hard to fully integrate our vis cluster support with the larger computational cluster support structure. Specifically:

- Cluster hardware is made as similar as possible
- Operating system is mostly identical
- Administration tools are identical
- Same pool of personnel handle the clusters

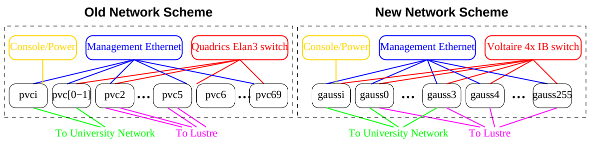
I/O — LLNL Site-wide Parallel Filesystem

LLNL vis clusters share Lustre filestores with the computational clusters, providing *in situ* access to simulation data. This in turn enables computational monitoring and computational steering without extensive code modification.



Scaling I/O Performance

After some initial dissatisfaction with I/O performance on early clusters, we worked with the LLNL Lustre team to re-think I/O for gauss, eliminating the gateway nodes and doing direct IP attachment.



As an added benefit, this means that per-node I/O bandwidth is constant as a job scales in size.

Unfortunately, it doesn't address Lustre QoS issues.

Job Management

LLNL vis clusters have an architecture similar to compute clusters, and thus can use the same scheduling and job initiation software.

- LCRM or Moab for site-wide job scheduling
- SLURM for resource management, job initiation, and cleanup
- X11 servers are started with jobs

As a result, the vis clusters are flexible, and post processing jobs that don't require X11 don't start it.

Anatomy of a Vis Job

```
psub -ln 8 -c gauss -b bdivp -tM 60m -x -i \
srun -N 8 /usr/bin/X11/xinit engine_par -- \
/usr/bin/X11/X
```

Meaning:

psub -ln 8 -c gauss -b bdivp -tM 60m -x -i: Submit a simple script to LCRM to run an 8 node job on gauss drawn on bdivp bank with the current environment exported to that job

srun -N 8 Ask SLURM to initiate the job on 8 nodes

/usr/bin/X11/xinit ??? /usr/bin/X11/X Start /usr/bin/X11/X and run the command ??? under it, when that command ends, terminate the X11 server

engine_par Our command (the parallel rendering engine from VisIt)

Job Management — Some Wins

- `xinit` and `startx` are pretty good about setting up some basic X11 security
- Software can do most of the complicated stuff
- X servers are usually reaped correctly
- slurm handles X11 cleanup when reaping fails
- The slurm PAM module keeps users off batch nodes, mitigating many of the X11 security concerns
- No X overhead for non-X11 jobs

Job Management — Some Losses

- Users have still have trouble with xinit syntax
- Good luck getting it right with dual video cards
- Users like to do insecure and antisocial things
- X11 security still a mixed bag

Clustered Vis is Cost Effective

	prism	bgl	peloton SU
visTFLOPs per $million	**42.6**	6.12	11.06
Luster BW (MB/s) per $million	**9600**	1280	3000
Time to First Image (sec)	**218.5**	436.9	786.4

GPUs are Still a Win

Single Precision FLOPS

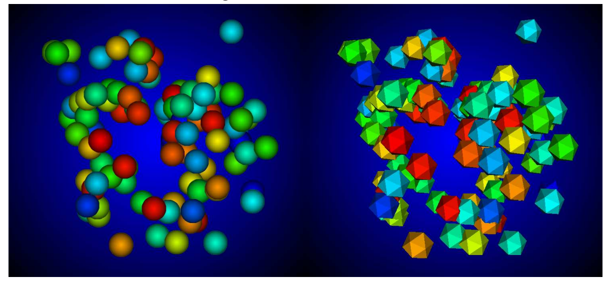
Opteron 280	**NVIDIA 7800GTX (G70)**
4 FLOPS/cycle	16 FLOPS/cycle
2 cores	24 pixel pipelines
2,400,000,000	430,000,000
19.2 GigaFLOPS	165 GigaFLOPS

Some hand-waving on both sides (Opteron won't be able to use SSE2 with transcendentals, G70 is counting the pixel shaders, not the vertex shaders).

Empirical experience: Parallel HW rendering is 6.3x faster than software rendering on a simple dataset.

GPUs are Really a Win

Skinning of Point Data: GPU vs CPU



Future is Remote Rendering

- Data is growing faster than networks, and desktops not getting better in all venues
- We have the GPUs
- We know how to scale clusters

So, multiple drivers for doing remote rendering. If only there was some middle-ware to enable it for general use....

Chromium RenderServer

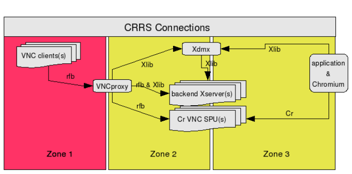
The Chromium Renderserver (CrRS) is software infrastructure that provides access to the virtual desktop on a remote computing system. It is a synthesis of several existing technologies:

- Xorg X11 and GLX for hardware support
- Chromium for distributed rendering of OpenGL
- DMX for distributed rendering of X11
- VNC (nee RFB) as a framebuffer transport protocol

As a result of the design, CrRS is completely open source and works with a variety of off-the-shelf viewer software. It is compatible with most OpenGL software and can be extended as necessary to meet site requirements.

CrRS Security Model — Zones

Because using existing protocols and software were design goals, the CrRS software picture can look a little complex. For easier discussion, we've split the components into zones based on how most sites will map them to hardware.

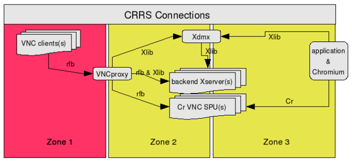


Is Zone 1 a Service or Not

- If the connection between the user and VNCproxy is a service like ftp or ssh, it should use the site authenticator (LDAP and OTP, NIS and crypt, etc.).
- If instead that connection is user traffic like MPI, we should insure that it's authorized via cookies or other means, but it probably should not go though authentication.
- In either case, both armoring and security can be implemented (eg, SSL and cookie, ssh and LDAP).

Outside of site-specific buzzword compliance, the question is more difficult.

Zones as Seen by LLNL



- Zone 2 and zone3 are treated like other backend cluster traffic. X11 security enabled in the provided launch scripts. The rfb traffic is authorized with a cookie distilled from the X11 cookie and only allows a single connection. No transport armor.
- Zone 1 is treated like traffic between two authenticated user processes. Traffic is SSL armored during transport, and underlying protocol (rfb) is configured to use a cookie that's distilled from the X11 cookie. The VNCproxy is also configured to disallow connection sharing and to exit on disconnect.

Discussion and Fighting